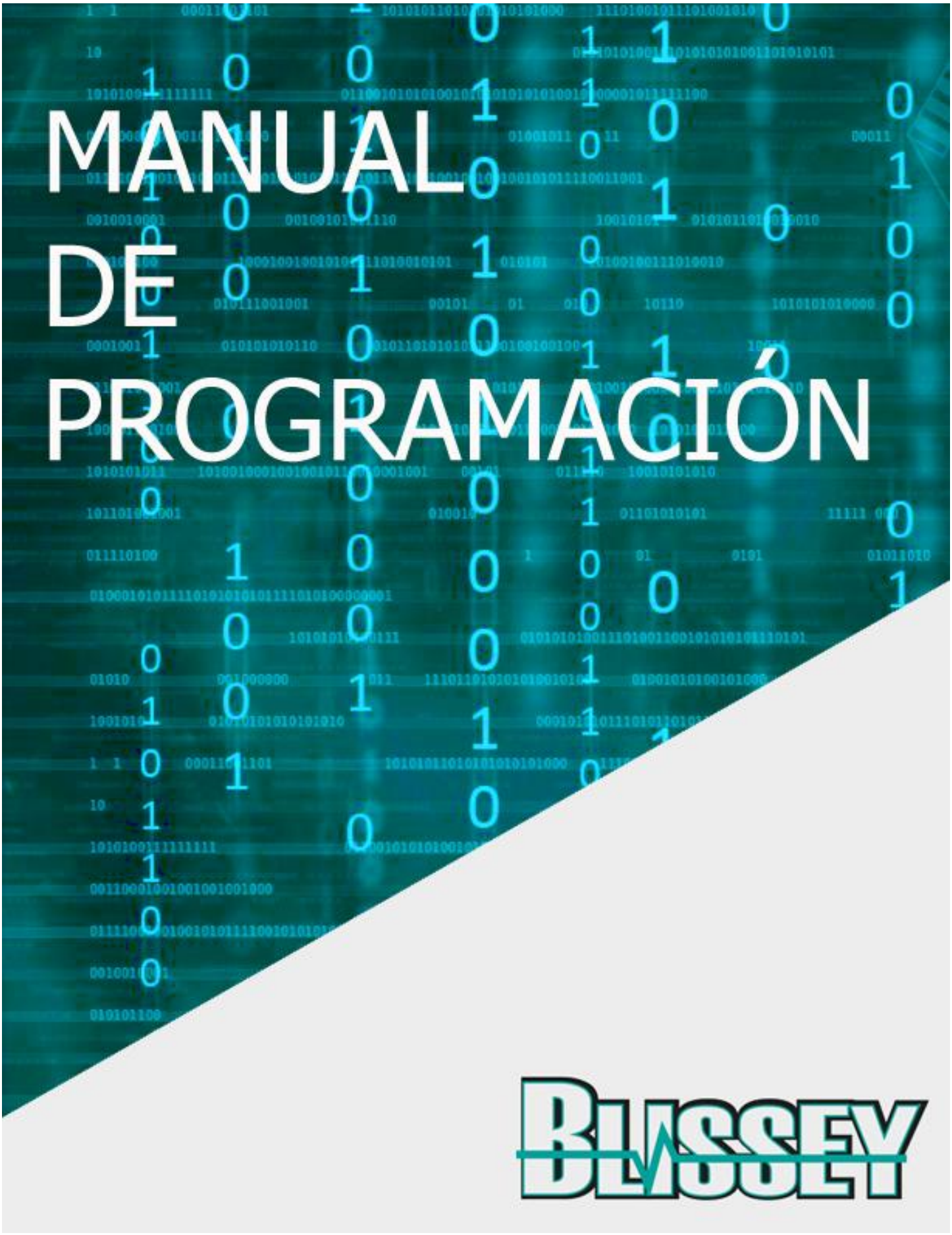
MANUAL DE PROGRAMACIÓN



Índice

Índice de figuras

Índice de tablas

Introducción

En el presente documento se detallan las principales bases para el desarrollo del Sistema Informático para la Administración de Áreas operativas del Grupo Promesa Divino Niño, en el municipio de San Vicente, departamento de San Vicente. a través de la programación contenida en el sistema se puede percibir una mejor idea del funcionamiento que este presenta.

Se detallan la codificación en general de las principales pantallas del sistema; estableciendo inicialmente el funcionamiento del framework laravel en su versión 5.4, método de descarga, conexión con base de datos y los diferentes comandos utilizados desde el símbolo del sistema para la creación de archivos, junto con su respectiva ubicación dentro del proyecto, también se muestra el manejo y cambios en la base de datos, ya sean estos adición, modificación o eliminación de campos.

También se detallan todos aquellos estándares que participan en la visualización de las pantallas, entre estos se encuentran el menú principal, menú superior junto con los diferentes campos que pueden componer los formularios; otro punto para tener en cuenta son las acciones que se pueden realizar sobre la información, por medio de las funciones crear, editar, activar, desactivar y eliminar un registro, procesos que pueden llevarse a cabo gracias a la codificación.

Objetivos

General

- Presentar los estándares de codificación contenidos en el Sistema Informático para la Administración de Áreas operativas del Grupo Promesa Divino Niño, en el municipio de San Vicente, departamento de San Vicente, por medio de una guía que facilite su contenido.

Específicos

- Detallar la estructura contenida en modelos, controladores y vistas que conforman en sistema.
- Identificar el código utilizado en la apariencia de menús, formularios y tablas de las vistas.

Laravel

Según la página oficial de Laravel se define como un marco de trabajo de código abierto, altamente funcional en la elaboración de aplicaciones y servicios web usando el lenguaje de programación php. Para la correcta utilización es necesario realizar las configuraciones básicas que se presentan a continuación:

Descarga de laravel.

Para iniciar la elaboración del sistema es necesario descargar el Framework Laravel a través del gestor de dependencias Composer, para ello realizar las siguientes instrucciones:

- Abrir el símbolo del sistema.
- Acceder al directorio de proyectos de XAMPP ingresando el siguiente comando: “cd c:/xampp/htdocs”, en caso de haber utilizado otra ruta debe modificar el comando.
- Instalar el proyecto ingresando los siguientes comandos: “composer create-project laravel/laravel blissey “5.4.*””, donde la palabra blissey representa el nombre del proyecto.
- Composer descargará el proyecto junto con todas las librerías necesarias.

Una vez finalice la descarga podemos confirmar el funcionamiento accediendo al proyecto desde el navegador con la ruta localhost/blissey/public, aparecerá una pantalla similar a la Figura 1.

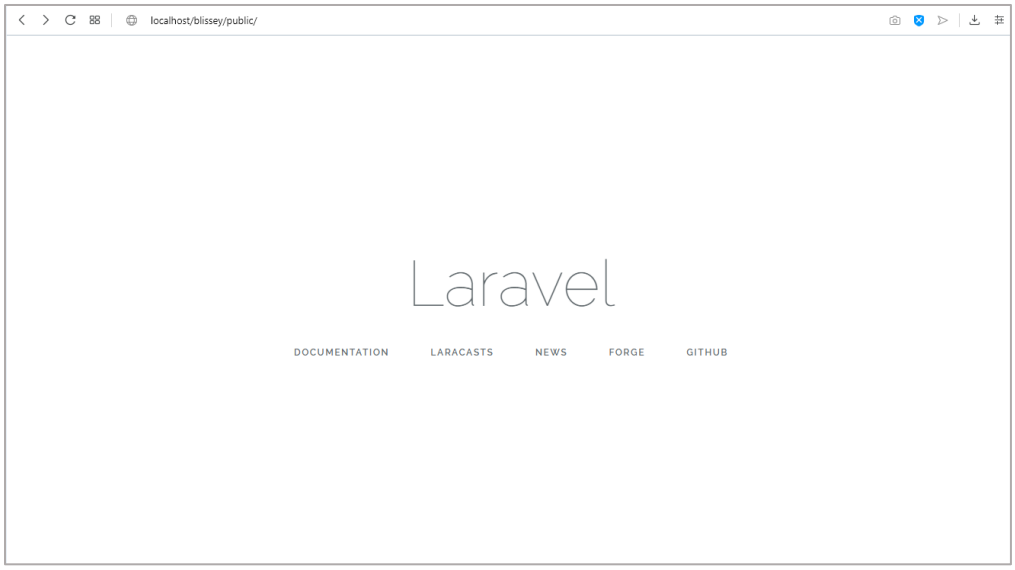


Figura 1. Pantalla principal Framework Laravel

Conexión con la base de datos

Al momento de crear el proyecto dentro debe aparecer un archivo .env.explample este archivo debe ser renombrando como .env, al abrir el archivo se verán ciertos datos incompletos debemos completar los datos que se detallan en la **¡Error! No se encuentra el origen de la referencia.**. En esta parte también se solicitan los datos del servidor del correo que serán utilizados para la recuperación de contraseña.

Tabla 1
Datos de conexión con la base de datos

Dato requerido	Función
DB_CONNECTION	Solicita el nombre del gestor de la base de datos: mysql
DB_HOST	El host para utilizar: localhost
DB_DATABASE	El nombre de la base de datos: blissey
DB_USERNAME	El nombre de usuario
DB_PASSWORD	La contraseña correspondiente al usuario

MAIL_DRIVER	El protocolo usado para el envío de correos: smtp
MAIL_HOST	El host usado para el envío de correo, en este caso hace uso del correo de Gmail: smtp.gmail.com
MAIL_PORT	El puerto usado para el envío: 567
MAIL_USERNAME	Correo creado en gmail
MAIL_PASSWORD	Contraseña generada en método de autenticación de dos pasos de gmail
MAIL_ENCRYPTION	Método de enctriptación para el envío: tls

Nota: Información obtenida de *https://laravel.com.mx/videos/fundamentos-laravel-5/libre/configuracion-y-.env*

Comandos por consola

Es necesario especificar ciertos comandos necesario para trabajar de forma correcta para ello es necesario ubicarse dentro de carpeta del proyecto que fue creado y desde ahí ejecutar los comandos a través de la interfaz de líneas de comando llamada Artisan. A continuación, se detallan los comandos a utilizar.

Comando key

Comando completo “php artisan key:generate”, cuando se descarga un proyecto este trae un código de seguridad que es necesario modificar para ellos ejecutamos el código mencionado desde el símbolo del sistema.

Comando make

Este comando permite crear una serie de archivos de forma automática, asignándolos en sus respectivas carpetas y con el código inicial. Ver algunos comandos de ejemplo en la Tabla 2.

Tabla 2
Comandos

Comando	Funcionamiento
php artisan make:controller ProductoController -r	Crea un nuevo controlador, el -r indica crear funciones por defecto.
php artisan make:model Producto -m	Crea un nuevo modelo, el -m indica crear una migración para el modelo.
php artisan make:migration table_cambios_caja	Crea una nueva migración, la cual permite modificar la base de datos.

Nota: Información obtenida de *https://styde.net/artisan-interfaz-linea-comandos-de-laravel/*

El comando make también es aplicable en request y middleware.

Comando migrate

Comando completo es "php artisan migrate", este comando se ejecuta cada vez que se realiza una modificación en la base de datos por medio de las migraciones ya sean estas para crear o eliminar tablas y campos. Es recomendable que cualquier modificación a la base de datos sea por medio de las migraciones y no directamente en el gestor de la base de datos.

Estándares de programación

Para el presente proyecto informático se ha hecho uso del Framework: Laravel versión 5.4, por lo que los estándares de programación utilizados son los que sugiere el Framework para su correcto uso. A continuación, se listan los elementos que posee. Ver Tabla 3.

Tabla 3
Estándares de programación

Elemento	Regla	Ejemplo
Modelo	El nombre se escribe con inicial mayúscula en singular, con extensión .php	Paciente.php
Controlador	Es el nombre del modelo, seguido de la palabra "Controller" haciendo uso de CamelCase, con	PacienteController.php

	extensión .php	
Directorios	Es el nombre del modelo en plural.	Pacientes
Vistas	El nombre va en minúsculas, con extensión .blade.php	index.blade.php
Migraciones	Es la palabra "create_table" seguido del nombre del modelo en minúsculas y plural (en inglés), con extensión .php	create_table_proveedors.php
Funciones	El nombre en con inicial minúscula haciendo uso de CamelCase	scopeNombre( )
Middleware	Es el nombre del tipo de usuario, seguido de la palabra "Middleware" en CamelCase, con extensión .php	RecepcionMiddleware.php
Scripts	Es el nombre del modelo en plural, con la extensión .js	Pacientes.js
Rutas	Es el nombre del modelo en plural (en español) y en minúsculas.	proveedores
Validadores	El nombre se escribe con inicial mayúscula en singular, seguido por la palabra "Request" con extensión .php	CajaRequest.blade.php

Nota: Información obtenida de *https://styde.net/convenciones-de-nombres-de-eloquent-en-laravel/*

Ubicación de elementos

Con el objeto de tener mejor control sobre la ubicación de los elementos Laravel presenta una estructura básica para almacenar todos los ficheros que sean creados dentro del proyecto ya sea por consola o de forma manual, ver ubicación en Tabla 4.

Tabla 4
Ubicación de elementos

Elemento	Ubicación
Modelo	/app
Controlador	/app/Http/Controllers
Directorios	/resources/views
Migraciones	/database/migrations

Middleware	/app/Http/Middleware
Scripts	/public
Archivo de rutas	/routes
Validadores	/app/Http/Requests

Nota: Información obtenida de *https://laravel.com/docs/5.4*

Archivo con rutas

Cuando se accede a una vista desde el navegador esta pasa primero a un controlado que se encarga obtener la información necesaria y redirigir al archivo solicitado, para que esto suceda es necesario declarar estas rutas en un archivo llamado web.php, existen diferentes formas para declarar rutas dependiendo de la funcionalidad y parámetros que reciba a continuación se presentan los casos posibles.

Ruta global

Una ruta global es aquella que permite acceder a las funciones básicas en el controlador sin necesidad de repetir una ruta por cada función. Se declara con el siguiente código: "Route::resource('transacciones','TransaccionController');". En la Tabla 5 se explica de manera específica cada elemento que conforma una ruta global.

Tabla 5
Elementos de una ruta global

Elemento	**Funcionamiento**
Route::resource();	Indica que la ruta es global por lo cual accede a las funciones predeterminadas, recibe como parámetros la ruta de acceso y el controlador asociado.
Parámetro 1 'transacciones'	Su nombre es asociado al controlador, representa la ruta de acceso mediante el navegador, generalmente se escribe en minúsculas y plural.

Parámetro 2 **'TransaccionController'**	Este parámetro es el nombre exacto del controlador al que está asociado.

Ruta específica

Es una ruta para acceder a aquellas funciones que se encuentran dentro del controlador pero que no son básicas y por lo tanto no están incluidas en la ruta global. Para declarar una ruta de este tipo hay muchas combinaciones que se pueden realizar de acuerdo con las necesidades que se tengan algunas combinaciones posibles son las siguientes:

- Route::match(['get','post'],'/buscarProductoTransaccion/{id}/{texto}','TransaccionController@buscarProductos');
- Route::get('/entradasver/{id}','CambioProductoController@ver');
- Route::post('/confirmarRetiroVencidos','CambioProductoController@confirmarRetiro');

En Tabla 6 se muestra los elementos que pueden conformar una ruta específica:

Tabla 6
Elementos de una ruta específica

Elemento	Funcionamiento
Route::match();	Esta clase de ruta recibe tres parámetros separados por comas, se usa para permitir el acceso a la ruta por medio de dos o más métodos
Route::get(); **Route::post();**	Esta clase de ruta recibe dos parámetros separados por una coma, permite el acceso por medio de un solo método indicado en el nombre, por lo cual ya no necesita especificarse.
Parámetro 1 **['get','post']**	Este parámetro contiene los métodos por los cuales se pude acceder a una ruta, disponible únicamente para rutas match.
Parámetro 2 **'/entradasver/{id}'**	Consta de dos partes: Una ruta de acceso con un nombre único y el paso de un parámetro denominado '/{id}', el cual se usa en rutas que lo requieran y puede repetirse para indicar el paso de otros parámetros

Parámetro 3 **'CambioProductoController@ver'**	Consta de dos partes: El nombre del controlador asociado y '@ver' que representa a la función creada en el controlador.

Modelo

En la lógica de codificación de Laravel, se llama modelo a la capa que tiene conexión con la base de datos, se debe crear un modelo por cada tabla en la base de datos, un ejemplo del código está en la Figura 2.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Reactivo extends Model
{
    protected $fillable = [
        'nombre', 'fechaVencimiento', 'contenidoPorEnvase',
    ];
    public static function buscar($nombre, $estado){…
    }

    public function scopeNombre($query, $nombre){…
    }

    public function scopeEstado($query, $estado){…
    }
}
```

Figura 2. Código del modelo.

Aquellas funciones que no serán invocadas desde los controladores necesitan el indicador "static" para funcionar sin problemas. Las funciones que contiene un modelo varían según la necesidad, entre estas están:

Variable fillable

Es un arreglo que contiene el nombre de los campos que podrán ser guardados desde la interfaz gráfica del usuario a la base de datos por tanto deben estar escritos de manera igual.

Función buscar

Esta función recibe uno o varios parámetros de búsqueda que son utilizados por los scope o en la consulta directamente. En la Figura 3 se muestra una función de búsqueda con un parámetro $estado.

```
12    public static function buscar($estado){
13      return Caja::estado($estado)->usuario()->orderBy('nombre')->get();
14    }
```

Figura 3. Código de la función buscar

Funciones scope

Son funciones que emplea Laravel para aplicar filtros y búsquedas en una tabla de la base de datos. Ver Figura 4, esta función recibe dos parámetros en la consulta, una variable $query y el parámetro para la búsqueda.

```
15    public function scopeEstado($query, $estado){
16      if($estado == null){
17        $estado = 1;
18      }
19      $query->where('estado',$estado);
20    }
```

Figura 4. Código de la función scope

Relaciones

Para facilitar las consultas a tablas asociadas en la base de datos, se emplean relaciones las cuales cambian según la cardinalidad de las tablas, estas pueden ser: hasMany, belongsTo, hasOne, belongsToMany, entre otras. En la Figura 5 se muestran distintas cardinalidades con los siguientes parámetros:

- Parámetro 1: Indica el namespace del proyecto “App”, acompañado por el modelo al cual hace referencia.
- Parámetro 2: Corresponde a la llave foránea que vincula el modelo del cual se invoca la función y el modelo que se quiere obtener.

```
15      public function habitacion(){
16        return $this->belongsTo('App\Habitacion', 'f_habitacion');
17      }
18
19      public function servicio(){
20        return $this->hasOne('App\Servicio', 'f_cama');
21      }
22
23      public function ingreso(){
24        return $this->hasMany('App\Ingreso', 'f_cama');
25      }
```

Figura 5. Código de funciones relacionales

Para hacer uso de cualquier función creada en un modelo independientemente de cuál sea solamente basta con instanciar el modelo. Ejemplo App\Cama::Buscar(1,0), o invocar con la ayuda de una variable perteneciente al modelo: $cama->buscar(1,1); en el caso de las funciones relacionales se usan con la variable de la siguiente forma: $cama->ingreso y se obtienen uno o varios registros dependiendo de la cardinalidad.

Controlador

Laravel como muchos Frameworks modernos posee una terminal llamada Artisan que permite generar algunas líneas de código.

Al hacer uso de Artisan para generar el código de un controlador, nos provee por defecto la estructura que aparecen en la *Figura **6***, en ella se muestran las instancias a ciertos modelos que son utilizados, la estructura del controlador y las funciones básicas.

```php
<?php

namespace App\Http\Controllers;

use App\Componente;
use Illuminate\Http\Request;
use App\Http\Requests\ComponenteRequest;
use Redirect;

class ComponenteController extends Controller
{
    public function index(Request $request)
    {···
    }
    public function create()
    {···
    }
    public function store(ComponenteRequest $request)
    {···
    }
    public function show(Componente $componente)
    {···
    }
    public function edit(Componente $componente)
    {···
    }
    public function update(Request $request, Componente $componente)
    {···
    }
    public function destroy($id)
    {···
    }
```

Figura 6. Estructura básica de un controlador generado con artisan

A continuación, se detallan cada una de las funciones:

Index

Es la función que se encarga de listar los elementos de una tabla. La función index no recibe parámetros por defecto, pero con agregar una variable $request como se puede observar en la Figura 7 la función recibirá todos los parámetros enviados a la función.

```php
public function index(Request $request)
{
  $pagina = ($request->get('page')!=null)?$request->get('page'):1;
  $pagina--;
  $pagina *= 10;
  $estado = $request->get('estado');
  $componentes = Componente::buscar($estado);
  $activos = Componente::where('estado',true)->count();
  $inactivos = Componente::where('estado',false)->count();
  return view('Componentes.index',compact(
    'componentes',
    'estado',
    'activos',
    'inactivos',
    'pagina'
  ));
}
```

Figura 7. Código de la función index

Luego de finalizar la búsqueda, la función index retorna a una vista los siguientes parámetros:

- Componentes: Los elementos de la tabla a retornar.
- Estado: El estado en base al cual se realizó la búsqueda en el modelo.
- Activos: Cantidad de activos encontrados.
- Inactivos: Cantidad de inactivos encontrados.
- Página: El número de paginación que se está listando en la vista.

Create

Con esta función se carga la vista para crear nuevos registros. En la Figura 8 se observa el código de la función create, este retorna al formulario donde se creará una nueva caja.

```php
public function create()
{
    return view('Cajas.create');
}
```

Figura 8. Código de función create

Store

Es la función encargada de almacenar en la base de datos lo que recibe del formulario create. En la Figura 9 se muestra como la función recibe por medio de $request toda información e instancia al model Reactivo para crear uno nuevo, para finalizar retorna a la vista principal junto con el respectivo mensaje.

```
47    public function store(ReactivoRequest $request)
48    {
49      Reactivo::create($request->All());
50      return redirect('/reactivos')->with('mensaje', '¡Guardado!');
51    }
```

Figura 9. Código de función store

Edit

Similar a create se encarga de mostrar la vista para editar registros. En la Figura 10 se muestra que la función recibe un identificador, con la ayuda de este, el registro es recuperado de la base de datos y enviado al formulario de edición.

```
81      public function edit($id)
82      {
83          $division=Division::find($id);
84          return view('Divisiones.edit',compact('division'));
85      }
```

Figura 10. Código de función edit

Update

Actúa de manera similar a store, la diferencia es que esta se encarga de almacenar los cambios a un registro en la base de datos. En la Figura 11 se muestra como antes de ser almacenado el registro se comprueban los cambios y valida que no sea un campo vacío. Luego retorna a la vista principal.

```php
public function update(Request $request,$id)
{
  $division=Division::find($id);
  if($request->nombre==$division->nombre){
    return redirect('/divisiones?estado'.$division->estado)->with('info', '¡No hay cambios!');
  }else{
    $validar['nombre']='required';
    $this->validate($request,$validar);
    $division->fill($request->all());
    $division->save();
    return redirect('/divisiones')->with('mensaje','¡Editado!');
  }
}
```

Figura 11. Código de función update

Show

Función utilizada para mostrar la información detallada de un registro. Como se muestra en la Figura 12 la función recibe como parámetro el identificador, este es buscado en base de datos junto a otros registros asociados por medio una relación, una vez obtenidos los datos necesarios estos son enviados a la vista.

```php
public function show($id)
{
    $paciente = Paciente::find($id);
    $ingresos = $paciente->ingreso;
    $solicitudes = $paciente->solicitudes;
    return view('Pacientes.show',compact(
      'paciente',
      'ingresos',
      'solicitudes'
    ));
}
```

Figura 12. Código de función show

Destroy

Función encargada de eliminar un registro de la base de datos. Esta función recibe el identificador perteneciente al registro que se desea eliminar, en la Figura 13 se puede observar que la función contiene una excepción, si esta falla indica que el registro contiene uno o más registros asociados en otras tablas y por lo tanto no podrá ser eliminado.

```php
public function destroy($id)
{
  try{
    $pacientes = Paciente::findOrFail($id);
    $pacientes->delete();
    return redirect('/pacientes?estado=0')->with('mensaje', 'Eliminado');;
  } catch (\Exception $e) {
    return redirect('/pacientes?estado=0')->with('error', 'Algo salio mal');
  }
}
```

Figura 13. Código de función destroy

Es importante aclarar que no es obligatorio usar las funciones antes mencionadas, pero hacen más sencillo el trabajo si se decide utilizarlas, así mismo es posible crear nuevas funciones dentro del controlador dependiendo de la necesidad.

Desactivate

Función encargada de cambiar el estado de un registro a inactivo. Esta función recibe el identificador perteneciente al registro que se desea enviar a inactivos, En la Figura 14 se puede observar que se busca el registro en la base de datos cambia su estado y actualiza el cambio luego redirecciona a la vista index.

```php
public function desactivate($id){
  $cajas = Caja::find($id);
  $cajas->estado = false;
  $cajas->save();
  Bitacora::bitacora('desactivate','cajas','cajas',$id);
  return Redirect::to('/cajas');
}
```

Figura 14. Código de función desactivate

Activate

Función encargada de cambiar el estado de un registro a activo. Esta función recibe el identificador perteneciente al registro que se desea enviar a activos, En la Figura 15 se puede observar que se busca el registro en la base de datos cambia su estado y actualiza el cambio luego redirecciona a la vista index de registros inactivos.

```
153        public function activate($id){
154            $cajas = Caja::find($id);
155            $cajas->estado = true;
156            $cajas->save();
157            Bitacora::bitacora('activate','cajas','cajas',$id);
158            return Redirect::to('/cajas?estado=0');
159        }
160    }
```

Figura 15. Código de función activate

Migración

En Laravel se simula una base de datos orientada a objetos, a diferencia de una base de datos relacional ordinaria, las tablas forman parte del código principal de la aplicación y a eso se le llama migración. Con las migraciones se crean las tablas relacionales en un gestor de base de datos ordinario, pero si se desea editar eso se debe crear una nueva migración para hacerlo.

Al igual que la creación de tablas, permite crear campos, en la Tabla 7 se muestran las sentencias para crear campos y otros usos.

Tabla 7
Tipo de campos disponibles

Tipo	Código	Uso
Variable identificadora	$table->increments('id');	Este campo representa una llave primaria tipo entero y autoincremental.
Cadena	$table->string('nombre',30);	Recibe el nombre de el campo y el tamaño máximo a ingresar
Número entero	$table->integer('indice');	Recibe el nombre de la variable
Número con punto decimal	$table->double('precio',8,2);	Recibe un nombre la cantidad de números máximos y la cantidad de decimales
Booleano	$table->boolean('activo');	Es para representar campos que solo pueden tener dos estados
Número sin signo	$table->integer('f_categoria')-	Un campo sin signo es creado

	>unsigned();	cuando se trabaja con llaves foráneas
Llave foránea	$table->foreign('f_categoria')->references('id')->on('categoria_servicios');	Luego de crear un campo para llave foránea es necesario indicar con que tabla está vinculada
Valor por defecto	$table->boolean('estado')->default(true);	Al momento de crear campos podemos asignar un valor por defecto.
Fecha	$table->date('fechaNacimiento');	Sirve para almacenar fechas dentro de la base de datos

Nota: Información obtenida de *https://laravel.com/docs/5.4/migrations*

Crear tabla

Artisan facilita la creación de migraciones mediante comando, un ejemplo de migración creada por comando se muestra en la Figura 16, la sentencia Schema::create dentro de la función up() indica que se creará una nueva tabla llamada reactivos y los componentes que están dentro de estos corresponden a los campos que se generarán en la tabla.

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateReactivosTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('reactivos', function (Blueprint $table) {
            $table->increments('id');
            $table->string('nombre');
            $table->string('descripcion');
            $table->string('contenidoPorEnvase');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('reactivos');
    }
}
```

Figura 16. Código de migración

Eliminar tabla

Para eliminar una tabla debe generarse una nueva migración, el código para eliminar una tabla se muestra en Figura 17, se usa la sentencia Schema::drop y se indica el nombre de la tabla a eliminar

```php
    public function up()
    {
      Schema::drop('nivels');
    }
```

Figura 17. Código para eliminar tabla

Agregar campos

Para realizar una nueva modificación en la cual se agregan nuevos campos, se necesita generar un nuevo archivo por comando, el código que corresponde a una modificación se puede ver en la Figura 18, esta función debe ubicarse dentro de up(), el cambio más notable es que la función cambia a Schema::table indicando que se realizan cambios en la tabla proveedors.

```
16          Schema::table('proveedors', function (Blueprint $table) {
17            $table->string('correo',30);
18            $table->string('telefono',9);
19            $table->boolean('estado')->default(true);
20          });
21      }
```

Figura 18. Código para agregar nuevos campos.

Eliminar campos

Para eliminar campos al igual que al agregar se necesita crear una migración por comando, en la Figura 19 se puede ver que usa la función Schema::table, se indica en que tabla se eliminarán los campos y cuales campos serán eliminados.

```
16          Schema::table('transacions', function (Blueprint $table) {
17              $table->dropColumn('localizacion');
18          });
```

Figura 19. Código para eliminar campos.

Vista

Al momento de elaborar la vista Laravel hace uso de Blade, que es una extensión similar a HTML 5 pero permite usar comandos para dibujar pantallas de forma más sencilla haciendo uso de templates y layouts. Mediante un modelo de dibujo por capas, permite códigos de vista más sencillos de leer y más ordenados.

Página principal

Para tener una mejor organización sobre el código que genera las vistas es necesario establecer un archivo principal.php dentro del directorio /resources/views, que se encargue de invocar las librerías agregadas en el directorio /public. Para un mejor aspecto en la vista, se hace uso de diferentes librerías y Frameworks de diseño como Bootstrap 4, JQuery, FontAwesome entre otras. la base de este código se muestra en la Figura 20.

```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <meta http-equiv="X-UA-Compatible" content="ie=edge">
7       <meta name="csrf-token" content="{{ csrf_token() }}">
8       <title>Blissey</title>
9       <input type="hidden" id="guardarruta" value="">
10      {{-- Dentro de head declarar archivos .css --}}
11  </head>
12  <body class="nav-md">
13    @endif
14    <div class="container-fluid h-100">
15      <div class="row h-100">
16        <div class="col-2 left_col side_back h-100 left-cont" style="overflow-x: hidden; overflow-y: scroll">
17          @include('Dashboard.panel_izquierdo')
18        </div>
19        <div class="col-10 side_back bg-light right-cont" role="main" style="overflow-x: hidden; overflow-y: scroll">
20          <div style="min-height:calc(100% - 51px)">
21            @yield('layout')
22          </div>
23          <div class="clearfix"></div>
24          <section>
25            <footer style="background-color: #DADADA">
26
27              <div class="footer-copyright text-center py-3">
28                UES - FMP 2018
29              </div>
30
31            </footer>
32          </section>
33        </div>
34        <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
35          {{ csrf_field() }}
36        </form>
37      </div>
38    </div>
39    {{-- Declarar archivos .js --}}
40  </body>
41  </html>
```

Figura 20. Código de vista principal

Dentro de las líneas de código contenidas en la Figura 20 es necesario detallar las siguientes:

- Línea 10: En este espacio se declaran las librerías con extensión .css.
- Línea 17: En esta parte del código se incluye el panel izquierdo de las pantallas.

- Línea 21: Cuando una vista extiende de principal.blade.php el código que contiene un section(‘layout’), será ubicado en esta parte del código.
- Línea 39: En este espacio se declaran las librerías con extensión .js.

Por cada entidad dentro del sistema es necesario especificar una serie de directorios por ejemplo para los usuarios debe crear el directorio /resources/views/Usuarios, dentro de este directorio se agregarán cuatro archivos con extensión .blade.php, los archivos se detallas a continuación:

Index

Es la pantalla donde se listan todos los elementos correspondientes a una entidad en la base de datos, que han sido enviados desde la función index de un determinado controlador, también se encarga enlazar otras opciones. El código de esta pantalla se muestra en la Figura 21 corresponde al index de unidades, este extiende de la pantalla principa.blade.php incluye su propia barra de opciones y recorre con la función foreach() todas las unidades.

Dependiendo del estado en que se encuentre la información solicitada muestra dos opciones “actívate” y “desactivate”.

```
@extends('principal')
@section('layout')
  @if ($estado == 1 || $estado == null)
    @php
    $estadoOpuesto = 0;
    @endphp
  @else
    @php
    $estadoOpuesto = 1;
    @endphp
  @endif
  @php
  $index = true;
  @endphp
  @include('Unidades.Barra.index')
  <div class="col-8">
    <div class="x_panel">
      <table class="table table-hover table-sm table-striped index-table">
        <thead>
          <tr>
            <th style="width: 30px;">#</th>
            <th>Nombre</th>
            <th style="width: 100px">Opciones</th>
          </tr>
        </thead>
        <tbody>
          @php
          $correlativo = 1;
          @endphp
          @foreach ($unidades as $unidad)
            <tr>
              <td>{{ $correlativo + $pagina }}</td>
              <td>
                {{ $unidad->nombre}}
              </td>
              <td>
                <center>
                  @if ($estadoOpuesto)
                    @include('Unidades.Formularios.activate')
                  @else
                    @include('Unidades.Formularios.desactivate')
                  @endif
                </center>
              </td>
            </tr>
            @php
            $correlativo++;
            @endphp
          @endforeach
        </tbody>
      </table>
    </div>
  </div>
  <!-- /page content -->
@endsection
```

Figura 21. Código de vista index

Opción desactivar.

Esta opción aparece disponible para aquellos registros que presente un estado activo. En este código se listan los enlaces disponibles para el registro. Estos se detallan a continuación:

- Form: Formulario que enlaza con una función del controlador compuesto por: ruta específica para desactivar, número identificador," método de envío".
- Opción ver: Es un enlace compuesto por: ruta global/número identificador del registro.
- Opción editar: Es un enlace compuesto por: ruta global/número identificador de registro/edit.
- Opción enviar a papelera: Se realiza mediante un botón y una función onclick con la ayuda de la librería sweetalert se declara un mensaje de confirmación, una vez este es confirmado se realiza submit en el form declarado.

En la Figura 22 se muestra el correspondiente código.

```
{!!Form::open(['url'=>['desactivateProducto',$producto->id],'method'=>'POST'])!!}
<div class="btn-group">
<a href={!! asset('/productos/'.$producto->id)!!} class="btn btn-sm btn-info" title="Ver">
  <i class="fas fa-info-circle"></i>
</a>
<a href={!! asset('/productos/'.$producto->id.'/edit')!!} class="btn btn-sm btn-primary" title="Editar">
  <i class="fas fa-edit"></i>
</a>
<button type="button" class="btn btn-danger btn-sm" title="Enviar a papelera" onclick="
  return swal({
    title: 'Enviar registro a papelera',
    text: '¿Está seguro? ¡Ya no estara disponible!',
    type: 'question',
    showCancelButton: true,
    confirmButtonText: 'Si, ¡Enviar!',
    cancelButtonText: 'No, ¡Cancelar!',
    confirmButtonClass: 'btn btn-danger',
    cancelButtonClass: 'btn btn-light',
    buttonsStyling: false
  }).then((result) => {
    if (result.value) {
      localStorage.setItem('msg','yes');
      submit();
    }
  });"/>
    <i class="fas fa-trash"></i>
</button>
</div>
{!!Form::close()!!}
```

Figura 22. Código opción desactivar

Opción activar.

Esta opción aparece disponible para aquellos registros que presente un estado inactivo. En este código se listan los enlaces disponibles para el registro. Estos se detallan a continuación:

- Form: Formulario que enlaza con una función del controlador compuesto por: método de envío y un id.
- Opción ver: Es un enlace compuesto por: ruta global/número identificador del registro.
- Opción editar: Es un enlace compuesto por: ruta global/número identificador de registro/edit.
- Opción restaurar: Se realiza mediante un botón y una función onclick que envía a una función alta() el valor identificador.
- Opción eliminar: Se realiza mediante un botón y una función onclick que envía a una función eliminar() el valor identificador. Se invoca una función en el modelo que recibe el identificador del registro, esta se encarga de verificar si el registro está relacionado en ese caso la opción eliminar aparece bloqueada. En la Figura 23 se muestra el correspondiente código.

```
{!!Form::open(['method'=>'POST','id'=>'formulario'])!!}
@if ($index)
<div class="btn-group">
  <a href={!! asset('/productos/'.$producto->id)!!} class="btn btn-sm btn-info"  title="Ver">
    <i class="fas fa-info-circle"></i>
  </a>
  <a href={!! asset('/productos/'.$producto->id.'/edit')!!} class="btn btn-sm btn-primary"  title="Editar">
    <i class="fas fa-edit"></i>
  </a>
</div>
<div class="btn-group">
  <button type="button" class="btn btn-success btn-sm"  title="Restaurar" onclick={!! "'alta(".$producto->id.");'" !!}>
    <i class="fas fa-check"></i>
  </button>
  @php
    $cuenta=App\Transacion::foraneos($producto->id);
  @endphp
  @if ($cuenta>0)
    <button type="button" class="btn btn-sm btn-danger disabled"  title="No se puede eliminar">
      <i class="fas fa-ban"></i>
    </button>
  @else
  <button type="button" class="btn btn-danger btn-sm"  title="Eliminar" onclick={!! "'eliminar(".$producto->id.");'" !!}>
    <i class="fas fa-times"></i>
  </button>
  @endif
</div>
@endif
{!!Form::close()!!}
```

Figura 23. Código opción activar

La función alta y eliminar deben ser declaradas en el archivo activate.blade.php, dentro de etiquetas javascript, abajo del código principal

Función alta

En la Figura 24 se muestra el código de una función alta(), por medio de esta se puede cambiar el campo estado de un registro, por tanto este aparecerá en la lista de registros activos, al igual que la opción enviar a papelera hace uso de la librería sweetalert, muestra un mensaje de confirmación y procede a asignar la ruta al form.

```
function alta(id){
  return swal({
    title: 'Restaurar registro',
    text: '¿Está seguro? ¡El registro estará activo nuevamente!',
    type: 'question',
    showCancelButton: true,
    confirmButtonText: 'Si, ¡Restaurar!',
    cancelButtonText: 'No, ¡Cancelar!',
    confirmButtonClass: 'btn btn-primary',
    cancelButtonClass: 'btn btn-light',
    buttonsStyling: false
  }).then((result) => {
    if (result.value) {
      localStorage.setItem('msg','yes');
      $('#formulario').attr('action','activateProducto/'+id);
      $('#formulario').submit();
    }
  });
}
```

Figura 24. Código función alta

Función eliminar

En la Figura 25 se muestra el código de una función eliminar(), por medio de esta se puede eliminar un registro de la base de datos bajo la condición de no existir una relación con otros registros, también muestra un mensaje de confirmación y procede a asignar la ruta al form.

```
function eliminar(id){
  return swal({
    title: 'Eliminar registro',
    text: '¿Está seguro? ¡El registro no podrá ser recuperado!',
    type: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#DD6B55',
    confirmButtonText: 'Si, ¡Eliminar!',
    cancelButtonText: 'No, ¡Cancelar!',
    confirmButtonClass: 'btn btn-danger',
    cancelButtonClass: 'btn btn-light',
    buttonsStyling: false
  }).then((result) => {
    if (result.value) {
      localStorage.setItem('msg','yes');
      $('#formulario').attr('action','destroyProducto/'+id);
      $('#formulario').submit();
    }
  });
}
</script>
```

Figura 25. Código función eliminar

Create

Esta pantalla es redirigida por la función create de un controlador, en ella se establece el método de envío del formulario hacia la función store del controlador al igual que se incluye en formulario con los correspondientes campos. En la Figura 26 se muestra el correspondiente código.

```
<?php $bandera=1;?>{{--Indica que es ingresar --}}
@extends('principal')
@section('layout')
  @include('Proveedores.Barra.create')
  {!!Form::open(['class' =>'form-horizontal form-label-left input_mask',
  'route' =>'proveedores.store','method' =>'POST','autocomplete'=>'off','id'=>'form'])!!}
      <div class="col-sm-12">
        @include('Proveedores.Formularios.form')
      </div>
      <input type="hidden" id="method" value="create">
  {!!Form::close()!!}
@endsection
@section('agregarjs')
{!!Html::script('js/scripts/Proveedores.js')!!}
```

Figura 26. Código de vista create

Formulario

Este incluye el código de todos los campos que componen el formulario. En la Figura 27 se muestra un formulario, este contiene al principio un mensaje con las instrucciones, también se presentan tres campos, cada uno agrupado junto a un label por medio de etiquetas.

```
<div class="alert alert-danger" id="mout">
  <center>
    <p class="mb-1">El campo marcado con un * es <b>obligatorio</b>.</p>
  </center>
</div>
<div class="x_panel">
  <div class="form-group">
    <label class="" for="nombre">Localización *</label>
    <div class="input-group mb-2 mr-sm-2">
      <div class="input-group-prepend">
        <div class="input-group-text"><i class="fas fa-list-alt"></i></div>
      </div>
      {!!Form::select('localizacion',$opciones,null, ['class'=>'form-control form-control-sm'])!!}
    </div>
  </div>
  <div class="form-group">
    <label class="" for="nombre">Código identificador *</label>
    <div class="input-group mb-2 mr-sm-2">
      <div class="input-group-prepend">
        <div class="input-group-text"><i class="fas fa-list-alt"></i></div>
      </div>
      {!! Form::number('codigo',$num,['class'=>'form-control form-control-sm','readonly'=>'readonly']) !!}
    </div>
  </div>
  <div class="form-group">
    <label class="" for="nombre">Número de niveles *</label>
    <div class="input-group mb-2 mr-sm-2">
      <div class="input-group-prepend">
        <div class="input-group-text"><i class="fas fa-list-alt"></i></div>
      </div>
      {!!Form::selectRange('cantidad', 1, 9,null,['class'=>'form-control form-control-sm'])!!}
    </div>
  </div>
</div>
<div class="x_panel">
  <center>
    {!! Form::submit('Guardar',['class'=>'btn btn-primary btn-sm']) !!}
    <button type="reset" name="button" class="btn btn-light btn-sm">Limpiar</button>
    <a href={!! asset('/estantes') !!} class="btn btn-light btn-sm">Cancelar</a>
  </center>
</div>
```

Figura 27. Código de formulario

Los campos de la Figura 27 no lo son los únicos campos que se pueden crear, a continuación, se detallan los distintos campos que son utilizados y sus parámetros:

Campo tipo select

El código de este campo se muestra en la Figura 28:

```
{!!Form::select('localizacion',$opciones,null, ['class'=>'form-control form-control-sm'])!!}
```

Figura 28. Código de un campo tipo select

Recibe los siguientes parámetros:

- Name: Nombre del campo.

- Array: Contiene las opciones a presentar dentro del select.
- Valor default: Es el índice de un valor seleccionado por defecto en caso de no poseer recibe el valor null.
- Array con otras especificaciones: Especificaciones como: class, id, readonly, etc.

Campo tipo number

El código de un campo se muestra en la Figura 29:

```
39 | | | {!! Form::number('codigo',$num,['class'=>'form-control form-control-sm','readonly'=>'readonly']) !!}
```

Figura 29. Código de un campo tipo number

Recibe los siguientes parámetros:

- Name: Nombre del campo.
- Valor default: Es un número dado por defecto en caso de no poseer recibe el valor null.
- Array con otras especificaciones: Especificaciones como: class, id, readonly, etc.

Campo tipo selectrange

El código tipo selectrange se muestra en la Figura 30:

```
48 | | | {!!Form::selectRange('cantidad', 1, 9,null,['class'=>'form-control form-control-sm'])!!}
```

Figura 30. Código de un campo tipo selectrange

Recibe los siguientes parámetros:

- Name: Nombre del campo.
- Valor inicial: Valor con que inicia el rango.
- Valor final: Valor con que termina el rango.
- Valor default: Es el índice de un valor del rango seleccionado por defecto en caso de no poseer recibe el valor null.

- Array con otras especificaciones: Especificaciones como: class, id, readonly, etc.

Campo tipo text

El código tipo text se muestra en la Figura 31:

```
14          {!! Form::text(
15            'nombre',
16            null,
17            ['class'=>'form-control form-control-sm',
18              'placeholder'=>'Nombre del usuario',
19              'id'=>'nombre_usuario_field']
20          ) !!}
```

Figura 31. Código de un campo tipo text

Recibe los siguientes parámetros:

- Name: Nombre del campo.
- Valor default: Es un texto dado por defecto en caso de no poseer recibe el valor null.
- Array con otras especificaciones: Especificaciones como: class, id, readonly, etc.

Campo tipo date

El código tipo date se muestra en la Figura 32:

```
94          {!! Form::date(
95            'fechaNacimiento',
96            $fecha,
97            ['class'=>'form-control form-control-sm',
98              'placeholder'=>'Nombre del paciente',
99              'max'=>$ahora->format('Y-m-d'),
100             'id' => 'fecha_paciente']
101         ) !!}
```

Figura 32. Código de un campo tipo date

Recibe los siguientes parámetros:

- Name: Nombre del campo.
- Valor default: Fecha asignada por defecto en caso de no poseer recibe el valor null.
- Array con otras especificaciones: Especificaciones como: class, id, readonly, etc.

Campo tipo textarea

El código tipo textarea se muestra en la Figura 33:

```
{!! Form::textarea(
  'direccion',
  null,
  ['class'=>'form-control form-control-sm',
    'placeholder'=>'Dirección del usuario',
    'rows'=>'2',
    'id'=>'direccion_usuario_field']
) !!}
```

Figura 33. Código de un campo tipo texarea

Recibe los siguientes parámetros:

- Name: Nombre del campo.
- Valor default: Es un texto dado por defecto en caso de no poseer recibe el valor null.
- Array con otras especificaciones: Especificaciones como: class, id, readonly, etc.

Campo tipo radiobutton

El código tipo radiobutton se muestra en la Figura 34:

```
<div id="radioBtn" class="btn-group col-sm-12">
  <a class="btn col-sm-6 btn-primary btn-sm active" data-toggle="sexo" data-title="1">Masculino</a>
  <a class="btn col-sm-6 btn-primary btn-sm notActive" data-toggle="sexo" data-title="0">Femenino</a>
</div>
```

Figura 34. Código de un campo tipo radiobutton

En el caso de un radiobutton no se usas las etiquetas existentes en html sino, se simula el mismo funcionamiento utilizando etiquetas <a>

Campo tipo file

El código tipo file se muestra en la Figura 35:

```
<input type="file" name="firma" class="custom-file-input" id="firma_file" lang="es">
```

Figura 35. Código de un campo tipo file

Corresponde a una etiqueta <input> con un type=”file” con el cual permite la subida de archivos, en este caso imágenes.

Edit

Esta pantalla es redirigida por la función edit de un controlador, que envía todos los datos correspondientes al registro que se desea editar, en ella se establece el método de envío del formulario hacia la función update del controlador al igual que se incluye un formulario, el cual contiene los datos ingresados al crear el registro. En la Figura 36 se muestra el correspondiente código.

```
1   <?php $bandera=2;
2   $id=$visitador->f_proveedor;
3   $estado=$visitador->estado;
4   ?>{{--Indica que es editar --}}
5   @extends('principal')
6   @section('layout')
7     @include('Visitadores.Barra.create')
8       {!! Form::model($visitador,['route'=>['visitadores.update',$visitador->id],'method'=>'PUT'
9       ,'class'=>'form-horizontal form-label-left input_mask','autocomplete'=>'off','id'=>'form']) !!}
10    <div class="col-sm-6">
11        @include('Visitadores.Formularios.form')
12    </div>
13    {!!Form::close()!!}
14  @endsection
15  @section('agregarjs')
16  {!!Html::script('js/scripts/Visitadores.js')!!}
17  @endsection
```

Figura 36. Código de vista edit

Show

Esta pantalla es redirigida por la función show de un controlador, que envía todos los datos correspondientes al registro que se desea mostrar, en la Figura 37 se observa el código, la pantalla se encuentra dividida en dos partes contenidas por etiquetas la primera con un tamaño de ocho columnas conteniendo un panel de navegación con dos opciones recuperadas por medio de una función relacional, este se encuentra desde la línea 8 a la 37 y la segunda división con un tamaño de cuatro columnas, donde se incluye otro archivo que muestra los datos

correspondientes al registro enviado desde la función show del controlador, se encuentra desde la línea 32 a la 34.

```
1  @extends('principal')
2  @section('layout')
3    @php
4      $index = false;
5      setlocale(LC_ALL,'es');
6    @endphp
7    @include('Productos.Barra.show')
8  <div class="col-sm-8">
9    <div class="x_panel">
10     <ul class="nav nav-tabs" id="myTab" role="tablist">
11       <li class="nav-item">
12         <a class="nav-link active" id="home-tab" data-toggle="tab" href="#home" role="tab" aria-controls="home" aria-selected="true">
13           Divisiones
14         </a>
15       </li>
16       <li class="nav-item">
17         <a class="nav-link" id="profile-tab" data-toggle="tab" href="#profile" role="tab" aria-controls="profile" aria-selected="false">
18           Componentes
19         </a>
20       </li>
21     </ul>
22     <div class="tab-content" id="myTabContent">
23       <div class="tab-pane fade show active" id="home" role="tabpanel" aria-labelledby="home-tab">
24         @include('Productos.Partes.datos_divisiones')
25       </div>
26       <div class="tab-pane fade" id="profile" role="tabpanel" aria-labelledby="profile-tab">
27         @include('Productos.Partes.datos_componentes')
28       </div>
29     </div>
30   </div>
31 </div>
32 <div class="col-sm-4">
33   @include('Productos.Partes.datos_producto')
34 </div>
35 <input type="hidden" id="id-p" value={{$producto->id}}>
36 @endsection
```

Figura 37. Código de vista show

En la Figura 38 se muestra el código correspondiente a un bloque del panel de navegación es el fragmento de código que se incluye en la línea 25 de la Figura 37, en este se recorre un objeto por medio de la función foreach().

```
1  <div class="row mt-2">
2    <div class="col">
3      <center>
4        <h5 class="mt-1">Componentes</h5>
5      </center>
6    </div>
7  </div>
8  <div class="ln_solid mt-3"></div>
9  <div class="row">
10   <div class="col-sm-12">
11     <table class="table table-hover table-sm table-striped index-table">
12       <thead>
13         <th>#</th>
14         <th>Componente</th>
15         <th>Contenido</th>
16       </thead>
17       @php
18         $contador_compoenente = 1;
19       @endphp
20       <tbody>
21           @foreach ($componentes as $componente)
22             <tr>
23               <td>{{$contador_compoenente}}</td>
24               <td>{{$componente->nombreComponente($componente->f_componente)}}</td>
25               <td>{{$componente->cantidad.' '.$componente->nombreUnidad($componente->f_unidad)}}</td>
26             </tr>
27             @php
28               $contador_compoenente++;
29             @endphp
30           @endforeach
31       </tbody>
32     </table>
33   </div>
34 </div>
```

Figura 38. Código bloque del panel de navegación.

Para mostrar la información correspondiente al registro que se desea ver se utiliza el código mostrado en la Figura 39**,** corresponde al fragmento de código que se incluye en la línea 33 de la Figura 37.

```
1  <div class="x_panel">
2    <div class="flex-row">
3      <center>
4        <h5 class="mb-1">Información General</h5>
5      </center>
6    </div>
7
8    <div class="ln_solid mb-1 mt-1"></div>
9    <div class="flex-row">
10     <span class="font-weight-light text-monospace">
11       Nombre
12     </span>
13   </div>
14   <div class="flex-row">
15     <h6 class="font-weight-bold">
16       {{$division->nombre}}
17     </h6>
18   </div>
19
20   <div class="ln_solid mb-1 mt-1"></div>
21   <div class="flex-row">
22     <span class="font-weight-light text-monospace">
23       Estado
24     </span>
25   </div>
26   <div class="flex-row">
27     <h6 class="font-weight-bold">
28       @if ($division->estado)
29         <span class="badge text-success border border-success col-4">Activo</span>
30       @else
31         <span class="badge text-danger border border-danger col-4">En papelera</span>
32       @endif
33     </h6>
34   </div>
35 </div>
```

Figura 39. Código bloque de información

Barra superior

En la parte superior de las vistas index, create, edit y show que se encuentran en los directorios dentro de /resources/view, se incluye un fragmento de código como se puede ver en la línea 15 de la Figura 21, este archivo recibe el mismo nombre que el archivo donde se incluye seguido de la extensión blade.php. En la Figura 40 se muestra el código de la barra superior, contiene el enlace que redirige hacia la vista index y el nombre, los cuales corresponden al título que aparecerá en la vista. A partir de la línea 20 se incluyen los distintos enlaces y opciones.

```
<nav class="navbar navbar-expand-lg navbar-light  sticky-top mb-2" style="background-color: #e3f2fd;">
  <a class="navbar-brand" href={!! asset('/divisiones') !!}>
    Divisiones
    @if ($bandera==1)
      <span class="badge border-success border text-success">
      Nuevo
    </span>
    @else
      <span class="badge border-purple border text-purple">
        Editar
      </span>
    @endif
  </a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavDropdown"
  aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNavDropdown">
    <ul class="navbar-nav mr-auto">
      {{-- Enlaces --}}
    </ul>
    @include('Dashboard.boton_salir')
  </div>
</nav>
```

Figura 40. Código barra superior

El código para agregar un enlace se presenta en la Figura 41, este ejemplo hace uso de una ruta global y redirige a la función create de un controlador.

```
<li class="nav-item">
  <a class="nav-link" href={!! asset('/divisiones/create') !!}>Nuevo</a>
</li>
```

Figura 41. Código enlace barra superior.

Otro tipo de opciones que se puede agregar en la barra superior son los menús desplegables en este caso se hace uso del código de la Figura 42, la etiqueta <a> de la línea 26 representa contiene el nombre e icono del menú y la segunda representa un enlace dentro del menú, esta etiqueta se repetirá por cada opción necesaria dentro del menú.

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink"
  role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
    Ver
  </a>
  <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
    <a class="dropdown-item" href={!! asset('/divisiones?estado='.$estadoOpuesto) !!}>
        Activos
        <span class="badge badge-primary float-right">
          {{$activos}}
        </span>
    </a>
  </div>
</li>
```

Figura 42. Código de menú desplegable en la barra superior.

La barra superior también incluye el archivo boton_salir.blade.php que se encuentra en el directorio /views/dashboard/, el cual contiene otras especificaciones como notificaciones y la opción salir, en la Figura 43, se observa el correspondiente código.

```
<ul class="navbar-nav mr-2">
  @include('Dashboard.notificaciones')
</ul>
<button type="button" class="btn btn-outline-primary btn-sm" id="blissey-out">
  Salir
</button>
```

Figura 43. Código botón salir de la barra superior.

Notificaciones

Para las notificaciones de la barra superior es necesario dividirlas según el tipo de usuario por lo tanto en el directorio views/Dashboard se crea el archivo notificaciones.blade.php, en este se incluyen otros archivos clasificados por tipo de usuario tal como aparece en la Figura 44, estos se encuentran en /views/Dashboard/notificaciones.

```
@if (Auth::check())
  @include('Dashboard.notificaciones.recepcion')
  @include('Dashboard.notificaciones.laboratorio')
  @include('Dashboard.notificaciones.ultrasonografia')
  @include('Dashboard.notificaciones.rayosx')
  @include('Dashboard.notificaciones.farmacia')
@endif
```

Figura 44. Código para incluir notificaciones.

En la Figura 45 se muestra el código que corresponde al menú desplegable de las notificaciones, En la línea 15 se agregan todas las notificaciones que se quieran mostrar.

```
1  @if(Auth::user()->tipoUsuario == "Laboaratorio")
2    @php
3      $solicitudes= App\SolicitudExamen::where('estado','=',0)->orderBy('id','desc')->get();
4    @endphp
5    <li class="dropdown nav-item">
6      <a href="#" class="nav-link dropdown-toggle" data-toggle="dropdown" aria-expanded="false" aria-haspopup="true">
7        <i class="fas fa-bell"></i>
8        @if ($solicitudes->count() > 0)
9          <span class="badge badge-danger">{{$solicitudes->count()}}</span>
10       @else
11         <span class="badge badge-success">{{$solicitudes->count()}}</span>
12       @endif
13     </a>
14     <div class="dropdown-menu new-drop msg_list list-unstyled" aria-labelledby="navbarDropdownMenuLink">
15       {{-- Enlaces --}}
16     </div>
17   </li>
18 @endif
```

Figura 45. Código menú de notificaciones.

Una vez se elabora el menú, es necesario agregar todas las notificaciones que se necesitan, en la Figura 46 se muestra el código para una notificación, está condicionada para aparecer y debe repetirse por cada notificación que se desea, en ella se muestra el enlace al que redirecciona, el título de la notificación y el conteo, pueden aparecer otros datos que se consideren necesarios.

```
28          @if ($stock>0)
29            <a class="dropdown-item" href="{{asset('/stockTodos')}}">
30              <div class="flex-row">
31                <center>
32                  <span class="text-uppercase text-monospace font-weight-light">Inventario bajo</span>
33                  <span class="badge badge-danger">
34                    {{$conteostock}}
35                  </span>
36                </center>
37              </div>
38              <span class="message">
39                Existen <strong>{{$conteostock}} productos</strong>  bajo el stock mínimo
40              </span>
41            </a>
42            <div class="dropdown-divider"></div>
43          @endif
```

Figura 46. Código notificación

Panel izquierdo

El panel izquierdo muestra información del usuario que ha iniciado sesión y las opciones disponibles según el tipo de usuario que se encuentra utilizando el sistema. La ubicación de este

archivo es /views/Dashboard bajo el nombre de panel_izquierdo.blade.php, el cual es llamado desde la página principal. El código de este archivo se divide en dos secciones:

Información del usuario: Contiene la imagen del usuario en caso se haya proporcionado, en caso contrario muestra una imagen por defecto, un corto saludo y el nombre del usuario. En la Figura 47 se muestra el código correspondiente

```
<div class="left_col scroll-view">

  <div class="navbar nav_title sticky-top" style="border: 0;">
    <a href={{asset( '/')}}>
      <center>
        <img src={{asset('img/blissey1.svg')}} alt="" style="width: 55%;">
      </center>
    </a>
  </div>

  <img src={!! asset(Storage::url((Auth::check())?Auth::user()->foto:"NoImgen.jpg")) !!} alt="" class="image-porfile">

  <center>
    <span>
        @if (Auth::check())
          {{(Auth::user()->sexo)?"Bienvenido":"Bienvenida"}}
        @endif
    </span>
  </center>

  <center>
    <spna class="font-sm">
      @if (Auth::check())
        <a href={{asset( '/usuarios/'.Auth::user()->id)}} class="text-light text-uppercase" style="text-shadow: 1px 1px #000;">
          {{Auth::user()->nombre}}
        </a>
      @else
      @endif
    </spna>
  </center>

  <div class="ln_solid mt-1 mb-1"></div>
  <div class="clearfix"></div>
```

Figura 47. Panel izquierdo información del usuario

Menú principal: Contiene todas las opciones disponibles agrupadas según el tipo de usuario, en el caso de que se esté creado el primer usuario el menú se omitirá por medio de una condición en su lugar aparecerá un mensaje, tal como aparece en la *Figura **48***.

```
<div class="main_menu_side hidden-print main_menu" id="sidebar-menu">
  <div class="menu_section">
    @if (Auth::check())
      <ul class="nav side-menu d-block" style="">
        @if (Auth::user()->tipoUsuario == "Recepción")
          @include('Dashboard.menu_recepcion')
        @elseif(Auth::user()->tipoUsuario == "Laboaratorio")
          @include('Dashboard.menu_laboratorio')
        @elseif(Auth::user()->tipoUsuario == "Farmacia")
          @include('Dashboard.menu_farmacia')
        @elseif(Auth::user()->tipoUsuario == "Enfermería" || Auth::user()->tipoUsuario == "Médico" ||
        Auth::user()->tipoUsuario == "Gerencia")
          @include('Dashboard.menu_medico')
        @elseif(Auth::user()->tipoUsuario == "Ultrasonografía")
          @include('Dashboard.menu_ultra')
        @elseif(Auth::user()->tipoUsuario == "Rayos X")
          @include('Dashboard.menu_rayosx')
        @elseif(Auth::user()->tipoUsuario == "TAC")
          @include('Dashboard.menu_tac')
        @endif
        @if (Auth::user()->administrador)
          @include('Dashboard.menu_admin')
        @endif
        @include('Dashboard.menu_general')
      </ul>
    @else
      <center>
        <span class="badge badge-danger">
          Registro del primer usuario
        </span>
      </center>
    @endif
  </div>
</div>
```

Figura 48. Panel izquierdo inclusión de menú desplegable.

Menú desplegable.

En el menú desplegable aparecen todas las opciones disponibles al momento de iniciar sesión, estas opciones dependen del tipo de usuario que inició sesión por tanto, en la *Figura **48*** se muestra que la inclusión del archivo que contiene el menú está condicionada, estos archivos se encuentran en el mismo directorio /views/Dashboard su nombre está conformado por “menu_” seguido del tipo de usuario con extensión blade.php, en la *Figura **49*** aparece el código correspondiente a un menú desplegable, en donde se muestran las siguientes etiquetas:

- Etiqueta <li>: Son todas las opciones disponibles, cuenta con una etiqueta <a> la cual contiene el enlace y el nombre de la opción.
- Etiqueta <a>: También son utilizadas mostrar el nombre de una opción desplegable.

- Etiqueta <ul>: Agrupa enlaces para indicar que se trata de una opción desplegable, dentro de esta etiqueta se puede incluir otras del mismo tipo para agrupar hasta tres niveles.

```
<li>
  <a>
    <i class="fas fa-microscope"></i> Laboratorio Clínico
    <span class="fas float-right fa-chevron-down"></span>
  </a>
  <ul class="nav child_menu">
    <li>
      <a href={{asset( '/solicitudex')}}>Evaluación de exámenes</a>
    </li>
    <li>
      <a href={{asset( '/bancosangre')}}>Banco de sangre</a>
    </li>
    <li>
      <a>Mantenimiento
        <span class="fas float-right fa-chevron-down"></span>
      </a>
      <ul class="nav child_menu">
        <li>
          <a href={{asset( '/examenes')}}>Exámenes</a>
        </li>
        <li>
          <a href={{asset( '/reactivos')}}>Reactivos</a>
        </li>
        <li>
          <a href={{asset( '/parametros')}}>Parámetros</a>
        </li>
        <li>
          <a href={{asset( '/secciones')}}>Tipo de secciones</a>
        </li>
        <li>
          <a href={{asset( '/muestras')}}>Tipo de muestras</a>
        </li>
        <li>
          <a href={{asset( '/unidades')}}>Unidades de medida</a>
        </li>
      </ul>
    </li>
  </ul>
</li>
```

Figura 49. Menú desplegable

Sesiones de usuario

Para las sesiones de usuario, Laravel cuenta con su propio mecanismo de autenticación llamado Middleware el cual permite y restringe el paso aciertas rutas.

Inicio de sesión

Para el inicio de sesión es debe crear en el directorio /views/auth el archivo login.blade.php su código aparece en la *Figura **50***, lo principal en esta estructura son los campos que solicitan el usuario y contraseña, los datos se envían por método "post" a la ruta "authenticate"

```
<!DOCTYPE html>
<html lang="es">
  <head>
{{-- css --}}
  </head>
  <body class="login">
    <div>
      <a class="hiddenanchor" id="signup"></a>
      <a class="hiddenanchor" id="signin"></a>
      <div class="login_wrapper">
        <div class="animate form login_form">
          <section class="login_content">
            <form method="POST" action="{{ route('authenticate')}}" autocomplete="off">
              {{ csrf_field()}}
              <h1>Bienvenido</h1>
              <div>
                {!! Form::text('name',null,['class'=>'form-control','placeholder'=>'Usuario','required'=>'',
                'autofocus'=>''])!!}
              </div>
              <div>
                {!! Form::password('password',['class'=>'form-control','placeholder'=>'Contraseña','required'=>''])!!}
              </div>
              <div>
                {!! Form::submit('Ingresar',['class'=>'btn btn-primary']) !!}
                <a class="reset_pass" href={{asset( '/password/email')}}>Olvidé mi contraseña</a>
              </div>
              <div class="clearfix"></div>
              <div class="separator">
                <div class="clearfix"></div>
                <br />
                <div>
                  <h1><i class="fa fa-medkit"></i> Blissey</h1>
                  <p>©2017 Universidad de El Salvador - Facultad Multidisciplinaria Paracentral</p>
                </div>
              </div>
            </form>
          </section>
        </div>
      </div>
    </div>
    <!-- jQuery -->
  </body>
</html>
```

Figura 50. Código de inicio de sesión.

Rutas autenticación

El siguiente paso es crear las rutas de acceso hacia las funciones de autenticación, las rutas que aparecen en la *Figura **51***, es necesario agregarlas en el archivo web.php

```
328    Route::get('/logout', 'Auth\LoginController@logout')->name('logout');
329    Route::get('/login', 'Auth\LoginController@login')->name('login');
330    Route::post('/authenticate', 'Auth\LoginController@authenticate')->name('authenticate');
```

Figura 51. Rutas de autenticación

Funciones de autenticación

Cuando se trabaja con laravel esté cuenta con un archivo de autenticación llamado LoginController.blade.php, que se encuentra ubicado en app/Http/Controller/Auth, es necesario agregar dentro del constructor del archivo las siguientes funciones:

Login: Esta función se encarga de redirigir las vistas, verifica por medio de la función Auth::check() si el usuario tiene sesión abierta, si esto es así, redirige a la página principal caso contrario redirige al inicio de sesión login. Ver *Figura **52***.

```
46      public function login()
47      {
48        if(Auth::check())
49        {
50          return redirect('/');
51        }
52        return view('auth.login');
53      }
```

Figura 52. Código función login

Authenticate: Esta función recibe la petición desde el login, compara el usuario y contraseña con los registros de la base de datos, si coinciden deja pasar a la vista principal caso contrarío redirige al login con un mensaje. Ver *Figura **53***.

```php
public function authenticate(Request $request)
{
    if (Auth::attempt(['name' => $request['name'], 'password' => $request['password'], 'estado' => true])) {
        Bitacora::bitacora('login','users','usuarios',Auth::user()->id);
        return redirect('/');
    }
    return redirect('login')->with('error', '¡El nombre de usuario o la contraseña son incorrectos!');
}
```

Figura 53. Código función authenticate

Logout: Cuando se llama esta función esta se encarga de cerrar la sesión activa y luego redirigir al login. Ver *Figura **54***.

```php
    public function logout()
    {
      if(Auth::check()){
        Bitacora::bitacora('logout','users','usuarios',Auth::user()->id);
      }
      Auth::logout();
      return redirect('login');
    }
}
```

Figura 54. Código función logout

Middleware

Los middleware son un mecanismo para filtrar peticiones de acceso a ciertas rutas, el primer paso es construir un middleware por medio del comando make, el código se muestra en la

*Figura **55***, cuenta una única función haddler que recibe la petición a la ruta luego compara si el usuario tiene una sección activa y también si es el tipo de usuario correcto es decir que el usuario tiene permitido el acceso a las rutas.

```php
<?php

namespace App\Http\Middleware;

use Closure;

class UsuarioLaboratorio
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
      if(auth()->check() && auth()->user()->tipoUsuario == 'Laboaratorio')
      {
        return $next($request);
      }
      return redirect("/");
    }
}
```

Figura 55. Código middleware

La segunda parte para el uso del middleware es agrupar las rutas a las que tiene permitido acceder el usuario, para ello utilizamos el código de la *Figura* ***56***, todos los usuarios que quedas acceder a dichas rutas y que no sean del tipo especificado retornarán a la pantalla principal.

```php
//Grupo laboratorio
Route::group(['middleware'=>'laboratorio'], function(){
  //Rutas de Reactivos
  Route::resource('reactivos','ReactivoController');
  Route::match(['get','post'],'/desactivateReactivo/{id}','ReactivoController@desactivate');
  Route::match(['get','post'],'/activateReactivo/{id}','ReactivoController@activate');
  Route::match(['get','post'],'/destroyReactivo/{id}','ReactivoController@destroy');
...
});
```

Figura 56. Código rutas agrupadas

Validación de campos

En este caso se usa la validación de campos mediante el método de validación Request, primero necesitamos generar un request mediante el método make una vez creado obtenemos un archivo que contiene las siguientes funciones:

Authorize: La cual retorna un valor, para que la validación funcione el valor que retorna debe ser "true".

Rules: Retorna un array que contiene el nombre del campo y las validaciones a aplicar sobre este separado por el símbolo "|".

Messages: Retorna un array con los mensajes a mostrar en cada situación indicándolo de la siguiente manera "campo_a_validar.validación"=>"Mensaje a mostrar".

El código correspondiente se muestra en la *Figura **57.***

Para hacer uso de la validación basta con instanciar desde la función store o update que recibe el formulario en un determinado controlador, el ejemplo se muestra en la siguiente línea:

"public function store(CategoriaProductoRequest $request)".

```php
class CajaRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
     public function rules()
     {
         return [
             'nombre'=>'required | min:5 | max:30 |unique:cajas',
             'localizacion'=>'required',
         ];
     }
     public function messages(){
        return [
             'caja.required'=>'El campo nombre es obligatorio',
             'caja.min'=>'El campo nombre necesita 5 caracteres mínimos',
             'caja.max'=>'El campo nombre necesita 30 caracteres máximo',
             'caja.unique'=>'El campo nombre ya ha sido registrado',

             'localizacion.required'=>'El campo localización es obligatorio',
        ];
     }
}
```

Figura 57. Código request

Los tipos de validación se muestran en la Tabla 8.

Tabla 8
Tipos de validación

Validación	Funcionamiento
Required	Indica que el campo es obligatorio
Min	El mínimo de caracteres que pueden ingresarse para una cadena, en caso de números el valor mínimo que puede ingresarse

Max	El máximo de caracteres que pueden ingresarse para una cadena, en caso de números el valor máximo que puede ingresarse
Unique	Que el valor no debe repetirse en una determinada tabla, para usar esto el campo en el formulario y la tabla deben ser exactamente iguales.
Alpha	Solamente se permite el ingreso de caracteres
Email	El dato ingresado debe corresponder a un correo electrónico.

Nota: Información obtenida de *https://laravel.com/docs/5.1/validation#available-validation-rules*

Una vez se envía el formulario las validaciones son aplicadas automáticamente, en caso de encontrar una regla no cumplida retorna a la vista del formulario un array con los mensajes de error especificados en la función messages(), para ver los errores es necesario ir directorio /resources/view y buscar principal.blade.php y añadir al final las líneas que aparecen en la *Figura **58***.

```
@foreach ($errors->all() as $error)
  @php
    echo '
    <script language="javascript">
      swal({
        toast: true,
        type: "error",
        title: "¡Error!",
        html: "'.$error.'",
        position: "top-end",
        showConfirmButton: false,
        timer: 4000
      });
    </script>
    ';
  @endphp
@endforeach
```

Figura 58. Código para mostrar errores de validación

Recuperación de contraseña

Para la recuperación de contraseña es necesario declarar en web.php la siguiente ruta:

Route::get('/password/email','Auth\ForgotPasswordController@form');

Esta ruta nos dará acceso a la función form declarada en ForgotPasswordController, archivo que fue creado automáticamente con el proyecto, el contenido de la función se muestra en *Figura **59***, la función retorna a un archivo en el directorio /views/auth/passwords, donde se debe crear el archivo email.blade.php

```
32      public function form()
33      {
34          return view('auth.passwords.email');
35      }
```

Figura 59. Función de retorno form.

El archivo debe contener el formulario que aparece en la *Figura **60***, donde la parte más importante es la ruta a la cual debe retornar "password.email". Con estos pasos el envío de mensaje de recuperación funciona usando la plantilla integrada por defecto.

```
35          <section class="login_content">
36            {!!Form::open(['route'=>'password.email','method'=>'POST','autocomplete'=>'off'])!!}
37              {{ csrf_field()}}
38              <h1>Recupera tu cuenta</h1>
39              <div>
40                {!! Form::text('email',null,['class'=>'form-control','placeholder'=>
41                'Correo electrónico','id'=>'email'])!!}
42              </div>
43              <div>
44                {!! Form::submit('Enviar',['class'=>'btn btn-primary']) !!}
45                <h5>Enviaremos un enlace a tu correo</h5>
46                <a class="reset_pass" href={{asset( '/')}}>Regresar</a>
47              </div>
48              {!! Form::close() !!}
49          </section>
```

Figura 60. Formulario para recuperación de contraseña

Para modificar el mensaje que envía al correo, debido a que el mensaje es en ingles se deben realizar los siguientes cambios:

Modificar el saludo y el cuerpo del mensaje ir al siguiente directorio /App/Notifications/ dentro se encuentra una archivo llamado MyResetPasword.php identificar la función toMail() y sustituir por las líneas que aparecen en la *Figura* ***61***.

```
25      public function toMail($notifiable)
26      {
27        return (new MailMessage)
28                  ->subject('Recuperar contraseña')
29                  ->greeting('Hola')
30                  ->line('Estás recibiendo este correo porque hiciste una solicitud de recuperación de
31                  contraseña para tu cuenta.')
32                  ->action('Recuperar contraseña', route('password.reset', $this->token))
33                  ->line('Si no realizaste esta solicitud, no se requiere realizar ninguna otra acción.')
34                  ->salutation('Saludos, Blissey');
35      }
```

Figura 61. Personalización de correo

Modificar el encabezado del mensaje ir al directorio /resources/views/verdor/mail/html y buscar header.blade.php sustituir la palabra “Laravel” por “blissey”.

Para finalizar ir a /resources/views/vendor/notifications y buscar email.blade.php, modificar el mensaje del enlace auxiliar por el que se muestra en la *Figura* ***62***.

```
53    @component('mail::subcopy')
54    Si tiene problemas para hacer clic en el botón "{{ $actionText }}", copie y pegue la
55    siguiente URL en su navegador web: [{{ $actionUrl }}]({{ $actionUrl }})
56    @endcomponent
```

Figura 62. Mensaje auxiliar.